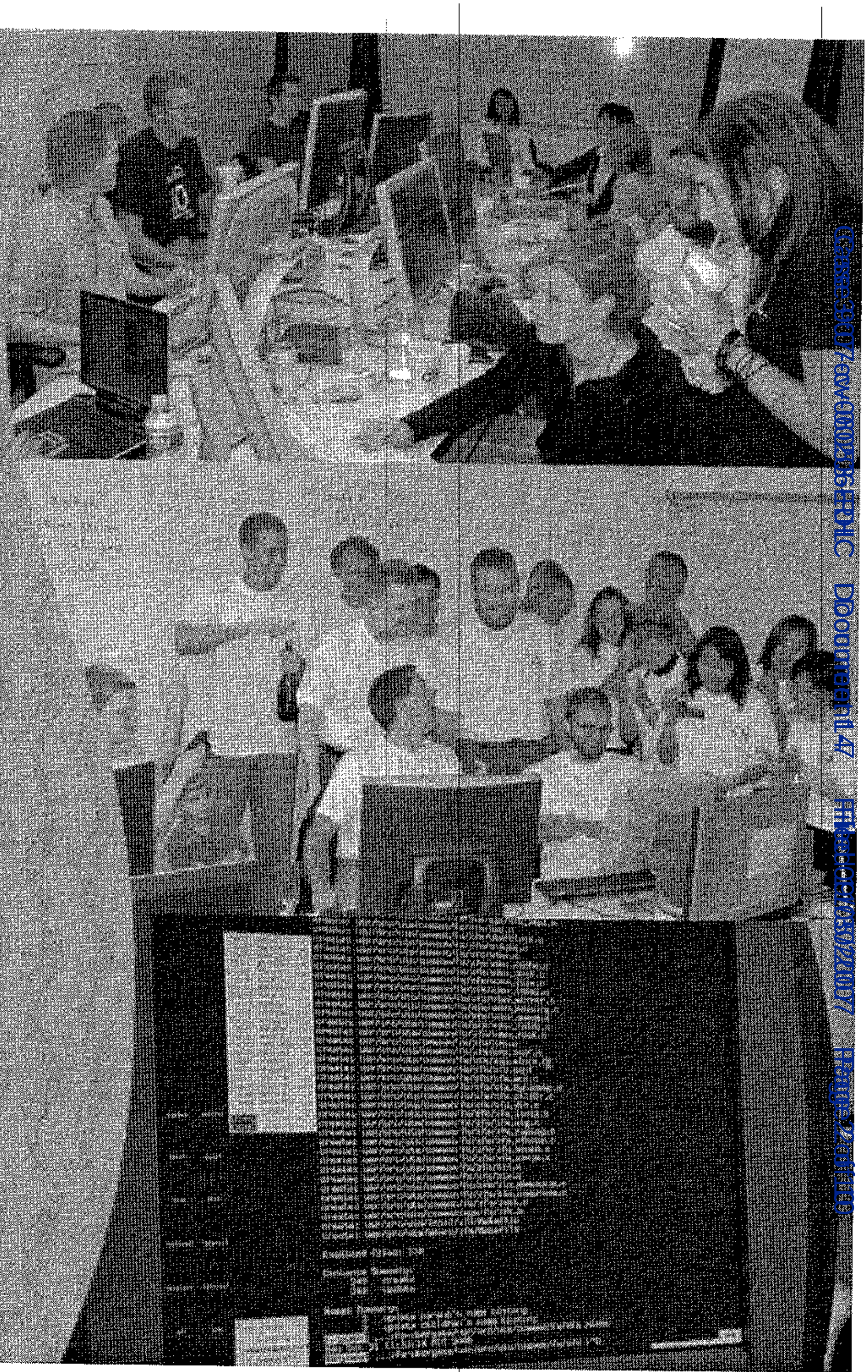
C

ZFS

THE LAST WORD IN FILE SYSTEMS

eric kustarz

www.opensolaris.org/os/community/zfs



ZFS Objective

End the Suffering

- Figure out why storage has gotten so complicated
- Blow away 20 years of obsolete assumptions
- Design an integrated system from scratch

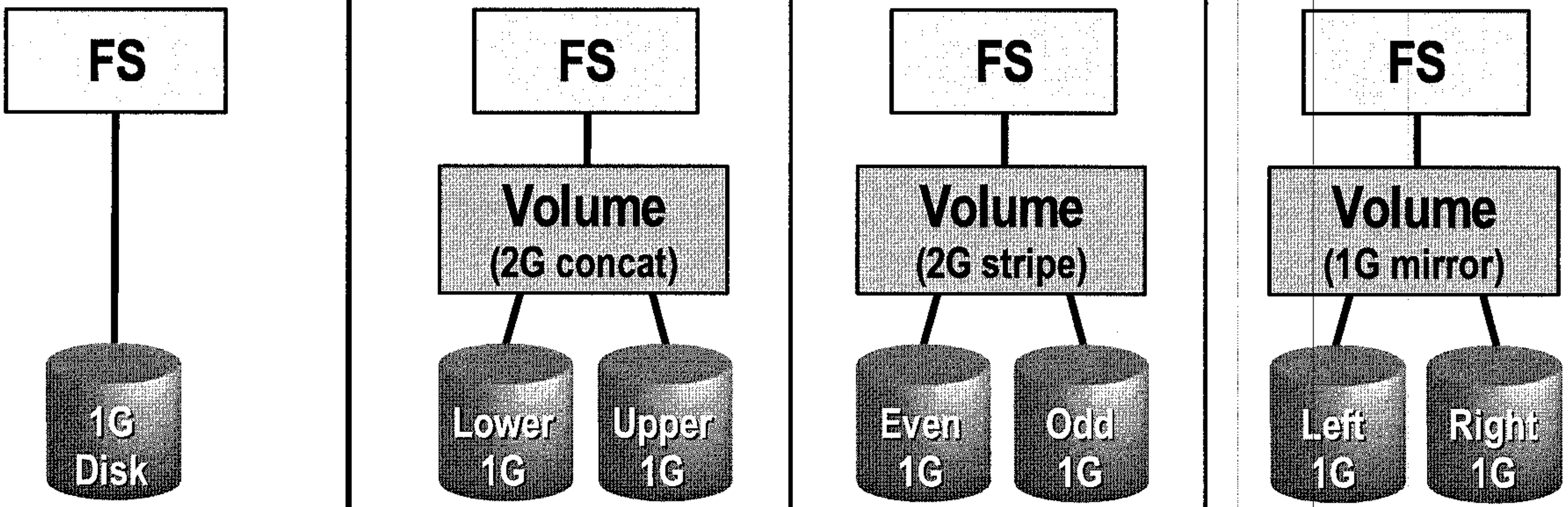
ZFS Overview

- Pooled storage
  - Completely eliminates the antique notion of volumes
  - Does for storage what VM did for memory
- Provable end-to-end data integrity
  - Detects and corrects silent data corruption
  - Historically considered “too expensive”
- Transactional design
  - Always consistent on disk
  - Removes most constraints on I/O order – huge performance wins
- Simple administration
  - Concisely express your intent

Why Volumes Exist

In the beginning, each filesystem managed a single disk.

- Customers wanted more space, bandwidth, reliability
  - Hard: redesign filesystems to solve these problems well
  - Easy: insert a little shim ("volume") to cobble disks together
- An industry grew up around the FS/volume model
  - Filesystems, volume managers sold as separate products
  - Inherent problems in FS/volume <u>interface</u> can't be fixed



FS/Volume Model vs. ZFS

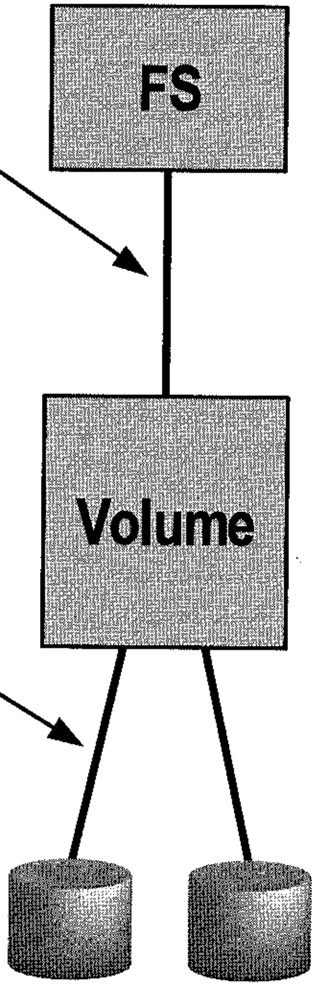
FS/Volume I/O Stack

Block Device Interface

- "Write this block, then that block, ..."
- Loss of power = loss of on-disk consistency
- Workaround: journaling, which is slow & complex

Block Device Interface

- Write each block to each disk immediately to keep mirrors in sync
- Loss of power = resync
- Synchronous and slow



ZFS I/O Stack

Object-Based Transactions

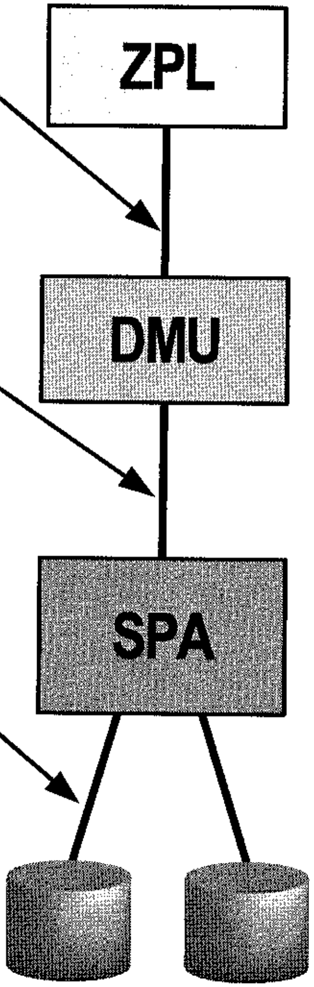
- "Make these 7 changes to these 3 objects"
- All-or-nothing

Transaction Group Commit

- Again, all-or-nothing
- Always consistent on disk
- No journal – not needed

Transaction Group Batch I/O

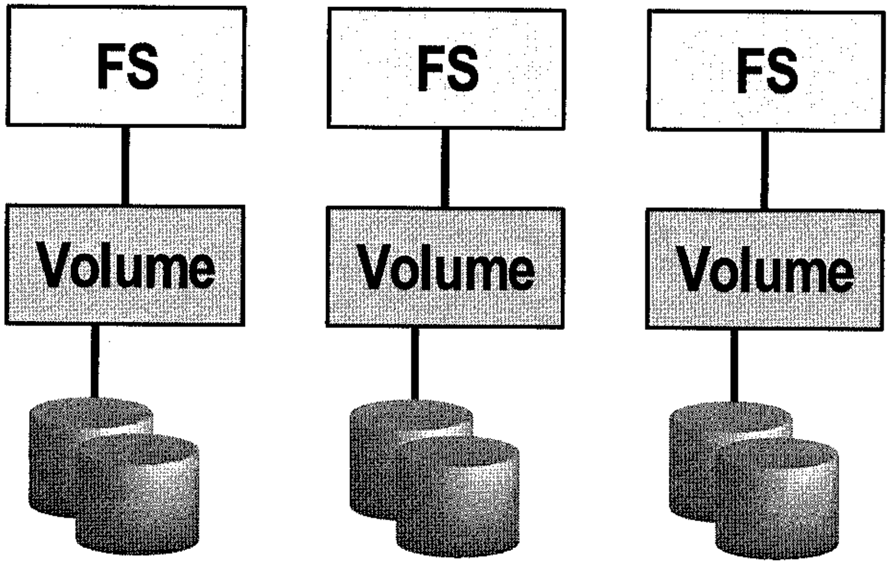
- Schedule, aggregate, and issue I/O at will
- No resync if power lost
- Runs at platter speed



FS/Volume Model vs. ZFS

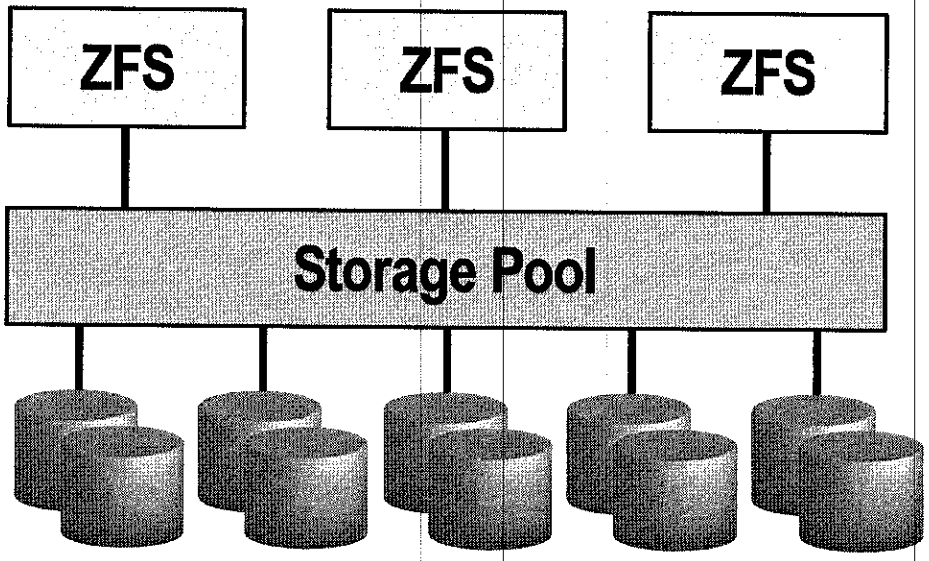
Traditional Volumes

- Abstraction: virtual disk
- Partition/volume for each FS
- Grow/shrink by hand
- Each FS has limited bandwidth
- Storage is fragmented, stranded

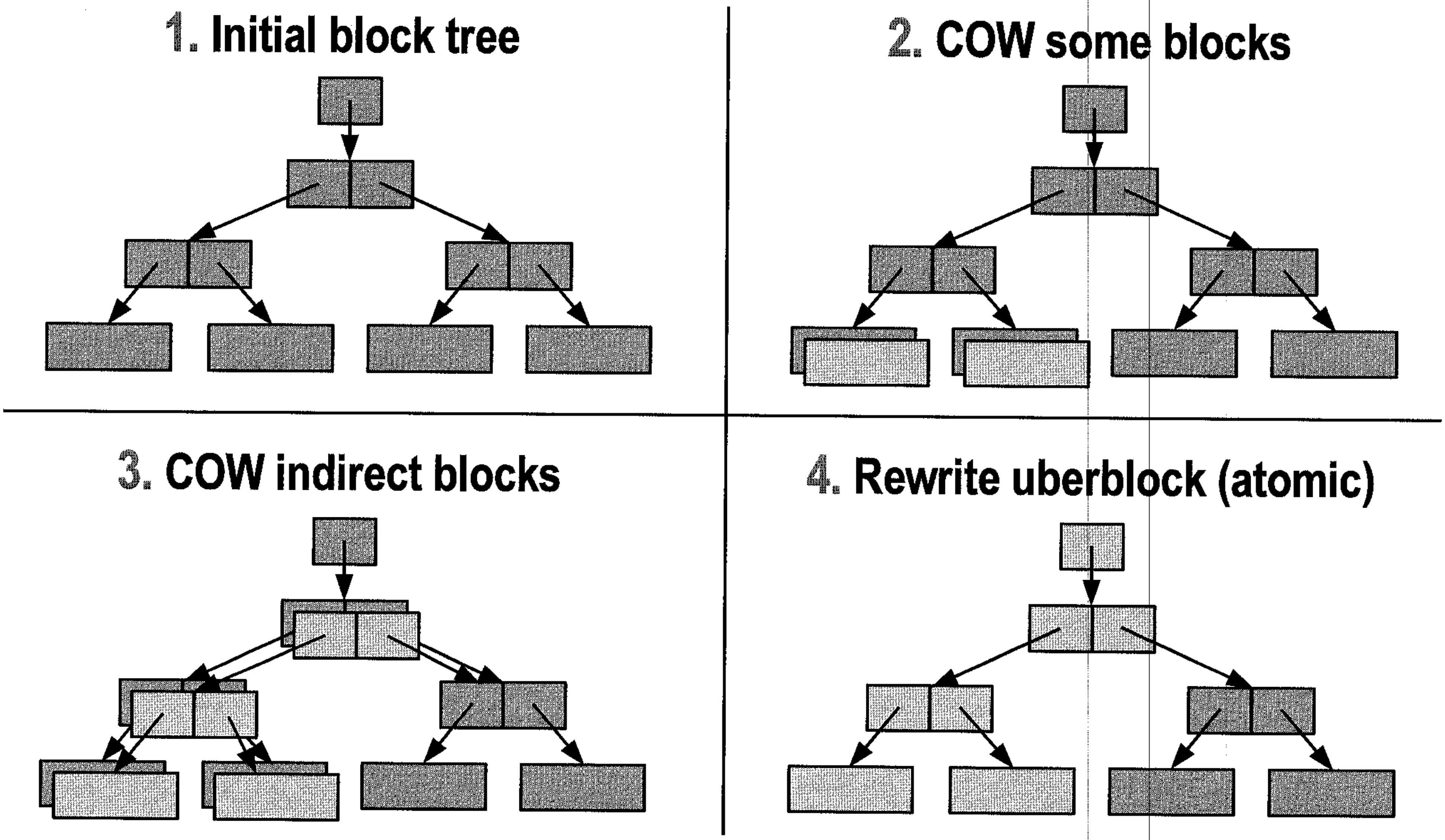


ZFS Pooled Storage

- Abstraction: malloc/free
- No partitions to manage
- Grow/shrink automatically
- All bandwidth always available
- All storage in the pool is shared



Copy-On-Write Transactions



1. Initial block tree

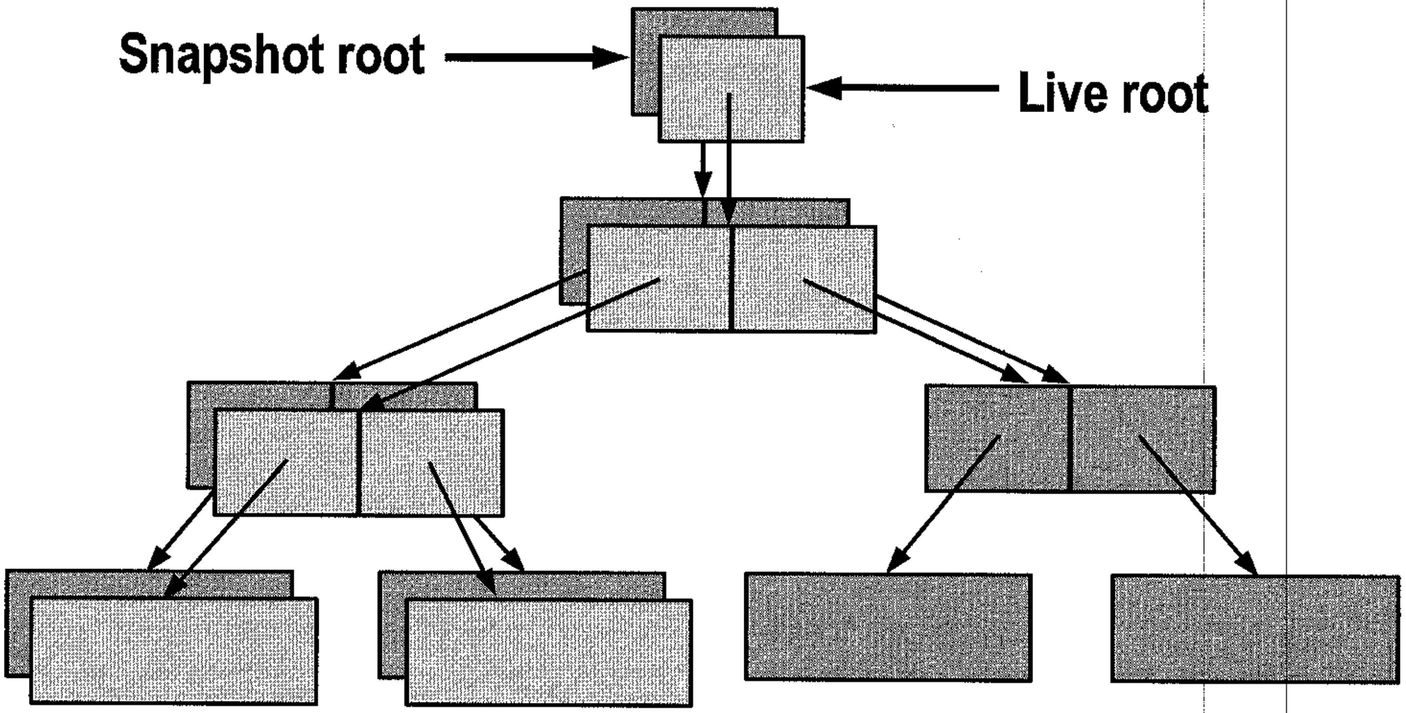
2. COW some blocks

3. COW indirect blocks

4. Rewrite uberblock (atomic)

Bonus: Constant-Time Snapshots

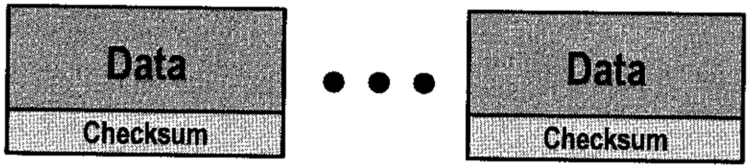
- At end of TX group, don't free COWed blocks
  - Actually cheaper to take a snapshot than not!



End-to-End Data Integrity

Disk Block Checksums

- Checksum stored with data block
- Any self-consistent block will pass
- Can't even detect stray writes
- Inherent FS/volume interface limitation

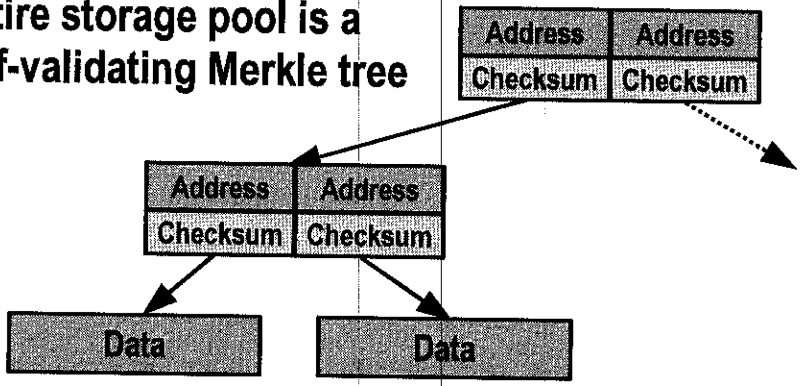


Disk checksum only validates media

- ✓ Bit rot
- ✗ Phantom writes
- ✗ Misdirected reads and writes
- ✗ DMA parity errors
- ✗ Driver bugs
- ✗ Accidental overwrite

ZFS Data Authentication

- Checksum stored in parent block pointer
- Fault isolation between data and checksum
- Entire storage pool is a self-validating Merkle tree



ZFS validates the entire I/O path

- ✓ Bit rot
- ✓ Phantom writes
- ✓ Misdirected reads and writes
- ✓ DMA parity errors
- ✓ Driver bugs
- ✓ Accidental overwrite